



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/085,455	02/27/2002	Motohiro Kawahito	JP920000420US1	1801

7590 06/18/2007  
ANNE V. DOUGHERTY  
3173 CEDAR RD.  
YORKTOWN HEIGHTS, NY 10598

EXAMINER

PHAM, CHRYSTINE

ART UNIT	PAPER NUMBER
----------	--------------

2192

MAIL DATE	DELIVERY MODE
-----------	---------------

06/18/2007

PAPER

**Please find below and/or attached an Office communication concerning this application or proceeding.**

The time period for reply, if any, is set in the attached communication.



UNITED STATES PATENT AND TRADEMARK OFFICE

---

Commissioner for Patents  
United States Patent and Trademark Office  
P.O. Box 1450  
Alexandria, VA 22313-1450  
[www.uspto.gov](http://www.uspto.gov)

**MAILED**

**JUN 18 2007**

**Technology Center 2100**

**BEFORE THE BOARD OF PATENT APPEALS  
AND INTERFERENCES**

Application Number: 10/085,455  
Filing Date: February 27, 2002  
Appellant(s): KAWAHITO ET AL.

Anne Vachon Dougherty  
Reg. No. 30,374  
For Appellant

**EXAMINER'S ANSWER**

This is in response to the appeal brief filed January 22, 2007 appealing from the Office action mailed July 26 2006.

**(1) Real Party in Interest**

A statement identifying by name the real party in interest is contained in the brief.

**(2) Related Appeals and Interferences**

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

**(3) Status of Claims**

The statement of the status of claims contained in the brief is correct.

**(4) Status of Amendments After Final**

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

**(5) Summary of Claimed Subject Matter**

The summary of claimed subject matter contained in the brief is correct.

**(6) Grounds of Rejection to be Reviewed on Appeal**

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

**(7) Claims Appendix**

The copy of the appealed claims contained in the Appendix to the brief is correct.

**(8) Evidence Relied Upon**

20020066086	Linden	05-2002
6760907	Shaylor	07-2004

**(9) Grounds of Rejection**

The following ground(s) of rejection are applicable to the appealed claims:

- Claims 1-4 and 6-16 are rejected under 35 USC 102(e) as anticipated by Linden.

**Claim 1**

*Linden* teaches a program optimization method (see at least *optimizing 14* FIG.2 & associated text; *14, 16* FIG.3 & associated text) for translating, into machine code (see at least *target instructions executable* paragraph [0003]), source code for a program written in a programming language (see at least *source instructions* paragraph [0003]), and for dynamically optimizing said program (see at least *dynamic compiler 10, optimizing 14* FIG.2 & associated text) comprising the steps of:

- performing a dynamic analysis during execution to determine whether the execution speed of said program can be increased by fixing, in a specific state, a parameter for a predetermined command in said program (see at least Abstract; *dynamically cross-compiling, execution speed, execution time, overhead* paragraphs [0010]-[0012]; *dynamic recompilation* paragraphs [0015]-[0016]; *decoded instruction, instruction sequence, result, constant, Register 3=0* paragraphs [0038], [0049]); and
- employing results of said analysis for the dynamic generation, in said program, of a path along which said parameter of said predetermined command is fixed in said specific state (see at least *instruction sequence, result, constant, Register 3=0, optimization step 44, optimized instruction stream* paragraph [0038]; see at least *decoding stage, optimization stage, flow of information, encoding stage* paragraph [0018]; paragraph [0037]; paragraph [0041]).

## Claim 2

The rejection of base claim 1 is incorporated. *Linden* further teaches wherein said step of generating a path includes the steps of:

- executing said program and obtaining statistical data for the appearance frequency of each available state (see at least *cross-compilation, execution loops of instructions* paragraph [0012];

paragraphs [0038]-[0039]) wherein, according to said results of said analysis, said parameter of said predetermined command may be set (see at least paragraphs [0038]-[0039]); and

- employing said obtained statistical data to dynamically generate said path (see at least *instruction sequence, result, constant, Register 3=0, optimization step 44, optimized instruction stream* paragraph [0038]; see at least *decoding stage, optimization stage, flow of information, encoding stage* paragraph [0018]; paragraph [0037]; paragraph [0041]).

### Claim 3

*Linden* teaches a program optimization method (see at least *optimizing 14* FIG.2 & associated text; *14, 16* FIG.3 & associated text), the source code for a program written in a programming language (see at least *source instructions* paragraph [0003]), and for optimizing said program comprising the steps of:

- executing a program to obtain statistical data for an appearance frequency of each available state in which a parameter of a predetermined command in said program may be set (see at least *cross-compilation, execution loops of instructions* paragraph [0012]; paragraphs [0038]-[0039]); and

- employing said obtained statistical data to dynamically generate a machine language program that includes, as the compiling results, a path (see at least *instruction sequence, result, constant, Register 3=0, optimization step 44, optimized instruction stream* paragraph [0038]; see at least *decoding stage, optimization stage, flow of information, encoding stage* paragraph [0018]; paragraph [0037]; paragraph [0041]).

#### **Claim 4**

The rejection of base claim 3 is incorporated. *Linden* further teaches comprising a step of: generating a machine language program that does not include, as a compiling result, a path along which said parameter of said predetermined command is fixed in a specific state (see at least *instruction sequence, result, constant* paragraph [0038]).

#### **Claim 6**

*Linden* teaches a program optimization method for translating, into machine code, the source code for a program written in a programming language, and for optimizing said program comprising the steps of:

- detecting dynamically during program execution one command, of the commands in said program, for which a variable can be limited to a predetermined constant value, and for which the processing

speed can be increased by limiting said variable to said constant value (see at least *instruction sequence, result, constant, Register 3=0* paragraph [0038]); and

- generating a path along which said constant value of said variable of said detected command is fixed (see at least *instruction sequence, result, constant, Register 3=0, optimization step 44, optimized instruction stream* paragraph [0038]).

#### Claim 7

*Linden* teaches a dynamic compiler (see at least *dynamic compiler 10, optimizing 14* FIG.2 & associated text) for translating into machine code the source code for a program written in a programming language (see at least *source instructions, target instructions executable* paragraph [0003]), and for optimizing the resultant program (see at least *dynamic compiler 10, optimizing 14* FIG.2 & associated text) comprising:

- an impact analysis unit for performing an analysis to dynamically determine during execution how much the execution speed of said program can be increased by fixing, in a specific state, a parameter of a predetermined command in said program (see at least *Abstract; dynamically cross-compiling, execution speed, execution time, overhead* paragraphs [0010]-[0012]; *dynamic recompilation* paragraphs [0015]-[0016]; *decoded instruction, instruction*



*sequence, result, constant, Register 3=0* paragraphs [0038], [0049]); and

- a specialization unit for employing the analysis results obtained by said impact analysis unit to generate, in said program, a specialized path along which said parameter of said predetermined command is fixed in said specific state (see at least *instruction sequence, result, constant, Register 3=0, optimization step 44, optimized instruction stream* paragraph [0038]).

#### **Claim 8**

The rejection of base claim 7 is incorporated. *Linden* further teaches:

- a data specialization selector for, when said program is executed, obtaining statistical data for the appearance frequency of each state obtained by said impact analysis unit, and for determining the state in which said parameter of said predetermined command is to be set (see at least *cross-compilation, execution loops of instructions* paragraph [0012]; paragraphs [0038]-[0039]),
- wherein said specialization unit generates a specialized path along which said parameter of said predetermined command is fixed in a state determined by said data specialization selector (see at least *instruction sequence, result, constant, Register 3=0, optimization*

*step 44, optimized instruction stream* paragraph [0038]; see at least *decoding stage, optimization stage, flow of information, encoding stage* paragraph [0018]; paragraph [0037]; paragraph [0041]).

### **Claim 9**

The rejection of base claim 8 is incorporated. *Linden* further teaches wherein, in accordance with the state of said program at execution, said specialization unit generates, in said program, a branching process for selectively performing a specialized path and an unspecialized path; and wherein, while taking into account a delay due to the insertion of said branching process, said data specialization selector determines a state in which said parameter of said predetermined command is fixed (see at least *optimized instruction flow stream, optimization rules, pipeline delay* paragraphs [0046]-[0047]).

### **Claim 10**

*Linden* teaches a computer (see at least FIG.1 & associated text) comprising:

- an input device for receiving source code for a program (see at least 42 Fig.3 & associated text);
- a dynamic compiler (see at least 10 FIG.2 & associated text) for translating said source code to compile said program and for

converting said compiled program into machine language code  
(see at least *source instructions*, *target instructions executable*  
paragraph [0003]); and

- a processor for executing said machine language code (see at least  
22 FIG.1 & associated text),
- wherein said dynamic compiler includes
- means for performing an dynamic analysis to determine during  
execution whether the execution speed of said program can be  
improved by fixing in a specific state a parameter of a  
predetermined command in said program (see at least Abstract;  
*dynamically cross-compiling*, *execution speed*, *execution time*,  
*overhead* paragraphs [0010]-[0012]; *dynamic recompilation*  
paragraphs [0015]-[0016]; *decoded instruction*, *instruction*  
*sequence*, *result*, *constant*, *Register 3=0* paragraphs [0038],  
[0049]), and
- means for generating in said program, based on the analysis  
results, a path along which said parameter of said predetermined  
command is fixed in said specific state and for compiling said  
program (see at least *instruction sequence*, *result*, *constant*,  
*Register 3=0*, *optimization step 44*, *optimized instruction stream*  
paragraph [0038]), and

- wherein said compiler outputs, as the compiled results, said machine language code that includes said path along which the state of said parameter is fixed (see at least *instruction sequence, result, constant, Register 3=0, optimization step 44, optimized instruction stream* paragraph [0038]; see at least *decoding stage, optimization stage, flow of information, encoding stage* paragraph [0018]; paragraph [0037]; paragraph [0041]).

#### **Claim 11**

*Linden* teaches a computer comprising:

- an input device, for receiving source code for a program (see at least 42 Fig.3 & associated text);
- a dynamic compiler (see at least 10 FIG.2 & associated text), for translating said source code to compile said program and for converting said compiled program into machine language code (see at least *source instructions, target instructions executable* paragraph [0003]); and
- a processor, for executing said machine language code (see at least 22 FIG.1 & associated text),
- wherein said dynamic compiler includes
- means for obtaining statistical data for the appearance frequency of each available state wherein a parameter for a predetermined

command in said program may be set when said program is executed, and for employing said statistical data to determine a state in which said parameter of said predetermined command is to be fixed (e.g., see *inductive expressions, multiplications, additions* col.4:1-10), and

- means for generating a specialized path along which said parameter of said predetermined command is fixed in said determined state, and for compiling said program (e.g., see *constant expression evaluation routine, runtime, object code image, Kfold routine* col.22:6-20; see *generating code* col.22:67-col.23:8; see *machine code, constant expression evaluation routine* col.23:32-35), and
- wherein said compiler outputs, as the compiled results, said program as said machine language code that includes said specialized path (e.g., see *constant expression evaluation routine, runtime, object code image, Kfold routine* col.22:6-20; see *generating code* col.22:67-col.23:8; see *machine code, constant expression evaluation routine* col.23:32-35).

#### **Claim 12**

The rejection of base claim 11 is incorporated. *Linden* further teaches comprising: said compiler further includes means for compiling

said program without generating a specialized path, wherein, when said state of said parameter to be fixed can not be determined, said means for determining the state of said parameter of said predetermined command outputs, as compiled results, said program in said machine language code, which is generated by said means for compiling said program without generating said specialized path, that does not include said specialized path (see at least *instruction sequence, result, constant* paragraph [0038]).

### **Claims 13-16**

Claims recite a computer medium containing a support program controlling a computer for performing the method, which have been addressed in claims 1-2, therefore, are rejected for the same reasons cited in claims 1-2.

- Claim 5 is rejected under 35 USC 103(a) as unpatentable over Linden in view of Shaylor.

### **Claim 5**

*Linden* teaches a program optimization method (see at least *optimizing 14* FIG.2 & associated text; *14, 16* FIG.3 & associated text) for translating, into machine code, the source code for a program written in a

programming language (see at least *source instructions, target instructions executable* paragraph [0003]).

*Linden* does not expressly disclose said programming language is an object-oriented programming language and said optimizing includes detecting one command dynamically during execution, of the commands in said program, for which a method call destination can be identified, and for which the processing speed can be increased by identifying said method call destination; and dynamically generating a path wherefor said method call destination for said detected command is limited in order to increase the processing speed of said command.

However, *Shaylor* discloses said programming language is an object-oriented programming language (see at least *Java source code 201* FIG.2 & associated text) and said optimizing includes detecting one command dynamically during execution (see at least *dynamic compiler 208* col.5:30-col.6:60), of the commands in said program, for which a method call destination can be identified (see at least "*method call*", "*inlining*" col.2:5-56), and for which the processing speed can be increased by identifying said method call destination (see at least "*method call*", "*inlining*" col.2:5-56; *optimization of native code, inlining techniques, method calls* col.4:5-42); and dynamically generating a path wherefor said method call destination for said detected command is limited in order to increase the processing speed of said command (see at least *method calls, types of*

*optimizations, inlining* col.6:50-62). *Linden* and *Shaylor* are analogous art because they are directed to dynamic compiler and optimization of executable code. It would have been obvious to one of ordinary skill in the art at the time of the invention to incorporate the teaching of *Shaylor* into that of *Linden* for the inclusion of object-oriented programming language and detecting and limiting method call destination. And the motivation for doing so would have been to enable optimization of execution speed for platform-independent programs (i.e., source code programs written in object-oriented programming language, such as Java source code) (see at least *Shaylor* col.1:20-45; col.2:48-56).

#### **(10) Response to Arguments**

Argument 1 (Brief, from page 13, 1st paragraph - top of page 14): *Linden* does not teach dynamically generating a path along which a *parameter* for a *predetermined command in said program* is *fixed in a specific state*. (Emphasis added)

Response to Argument 1: First as pointed out by Appellants (Brief, page 13), *Linden* is directed to a dynamic compiler and method in which source code is translated (i.e., dynamically compiled) to run (i.e., executable) on a target machine that is different from the machine for which the source code was developed. Second, as has been addressed in the Final Office Action dated 07/26/2006 (pages 2-3), in paragraph [0034] (associated with FIG.2), *Linden* explicitly discloses the dynamic compiler dynamically



decoding (i.e., performing a **dynamic analysis**) the source instructions and their parameters as a first step. The same passage explicitly discloses the **dynamic** compiler **creating/generating** an instruction stream (i.e., **path**) that is optimized **based on** said decoded (i.e., **predetermined**) **source instructions and parameters**. As established in Final Office Action, 44 of FIG.3, paragraphs [0037]-[0038] and [0041] of Linden explicitly discloses the **dynamic** decoding (i.e., **analysis**) stage going through block of source instructions and *analyzes* the operation codes such as the addition (i.e., **command**) of two numbers (i.e., **parameters**). Specifically in paragraph [0038], Linden explicitly discloses translating the assignment (i.e., command) of Register 3 (i.e., one of the parameters) to the outcome of "Register 1 XOR Register 1" which is always **fixed in a specific state** because clearly it does not matter what the value of variable Register 1 is, the result of XOR-ing a variable and itself is always the **constant 0** (i.e., Boolean false). In response to Appellants' argument that Linden creates new instructions for the target device "*independent of operations specified by the source instructions*" (Brief, page 13, first paragraph), it is respectfully submitted that the Appellants quoting of Linden is out of context because in paragraph [0041] (last 7 lines), Linden explicitly states that:

*"One can look at this optimization process as an interpolation from the source instructions to the equivalent results to be achieved by the target processor, independent of the operations specified by the source instructions, but dependent on the intended purpose and flow of the source instructions and how they are to be handled by the target processor to achieve an equivalent result."*

Thus, in this passage, Linden is merely offering a “bird’s-eye” view for looking at the **optimization process**, which in fact is **dependent on** the intended purpose of the **source instructions** and the instruction stream (i.e., path) generated from said source instructions during the decoding stage. Without considering the source instructions (and associated parameters), it is impossible to derive (i.e., generate) equivalent and optimized instructions to be executed by the target processor.

Argument 2 (Brief, page 14): *Eliminating operations* is not claimed and is not the same as the claimed step of determining whether execution speed can be increased by fixing a parameter for a command of the program in a specific state.

Response to Argument 2: The Examiner respectfully submits that although “eliminating operations” is not claimed, this feature of Linden is not excluded from the plain language of the claims. Furthermore, as addressed above in Response to Argument 1, paragraphs [0037]-[0038] and [0041] of Linden explicitly discloses the **dynamic** decoding (i.e., **analysis**) stage going through block of source instructions and *analyzes* the operation codes such as the addition (i.e., **command**) of two numbers (i.e., **parameters**). Specifically in paragraph [0038], Linden explicitly discloses translating the assignment (i.e., command) of Register 3 (i.e., one of the parameters) to the outcome of “Register 1 XOR Register 1” which is always **fixed in a specific state** because clearly it does not matter what the value of variable Register 1 is, the result of

Art Unit: 2192

XOR-ing a variable and itself is always the **constant 0** (i.e., Boolean false). In other words, Linden's decoding stage analyzes each instruction (i.e., command or operation) of the program to identify and "eliminate commands or operations that contain parameters which do not effect the operations" (i.e., fixed in a specific state). The purpose of doing this is to optimize the resulting code, which is well known in the art as the process making the program code (with less commands) executes more efficiently and/or rapidly (i.e., increase execution speed).

Argument 3 (Brief, pages 15-16): Linden does not teach "performing a dynamic analysis during execution to determine whether execution speed can be increased by fixing a parameter for a predetermined command in a specific state".

Response to Argument 3: As addressed above, in paragraph [0034] (associated with FIG.2), Linden explicitly discloses the dynamic compiler dynamically decoding (i.e., performing a **dynamic analysis**) the source instructions and their parameters as a first step. The same passage explicitly discloses the **dynamic** compiler **creating/generating** an instruction stream (i.e., **path**) that is optimized **based on** said decoded (i.e., **predetermined**) **source instructions and parameters**. As established in Final Office Action, paragraphs [0037]-[0038] and [0041] of Linden explicitly discloses the **dynamic** decoding (i.e., **analysis**) stage going through block of source instructions and *analyzes* the operation codes such as the addition (i.e., **command**) of two numbers (i.e., **parameters**). Also as discussed above, Linden's decoding stage (i.e., dynamic

Art Unit: 2192

analysis) analyzes each instruction (i.e., command or operation) of the program to identify and eliminate commands or operations that contain parameters which do not effect the operations (i.e., fixed in a specific state). The purpose of doing this is to optimize the resulting code, which is well known in the art as the process making the program code (with less commands) executes more rapidly (i.e., increase execution speed). Thus, contrary to Appellants' argument, Linden clearly anticipates 'performing the dynamic analysis'.

Argument 4 (Brief, pages 18-19): Linden does not teach "obtaining statistical data for the appearance frequency of each available state ... to dynamically generate the path".

Response to Argument 4: As established in the Final Office Action (page 6), paragraph [0012] of Linden explicitly discloses improving the over all execution speed by cross-compiling (also referred to as recompilation) programs containing *execution loops of instructions* that are repeatedly executed *hundreds, thousands of times* (i.e., appearance frequency of each available state) during execution of the program. The same paragraph also explicitly discloses analyzing and decoding these frequently executed instructions *once* to generate the target instruction in order to avoid repeated interpretation of the instructions in such loops, thereby saving (i.e., increasing) execution time/speed of the translated/compiled code (see also paragraphs [0016]). Thus, it is inherent in Linden that the frequency of being executed for each command is

determined (i.e., obtain statistical data for appearance frequency of each available state) in order to identify which command to dynamically recompile (as opposed to repeatedly interpreting the frequently executed command which would slow down the overall execution speed of the program) to generate the optimized instruction stream (i.e., path) for the target machine.

Argument 5 (Brief, page 21): Linden does not teach “a specialization unit which generates a branching process for selectively performing both a specialized path and an unspecialized path, or a data specialization selector for determining a state in which the parameter of a predetermined command is fixed”.

Response to Argument 5: As discussed above, paragraphs [0037]-[0038] and [0041] of Linden discloses the decoding stage (i.e., dynamic analysis) which analyzes each instruction (i.e., command or operation) of the program to identify and eliminate commands or operations that contain parameters which do not effect the operations (i.e., fixed in a specific state). Furthermore, as established in the Final Office Action (page 10), paragraphs [0046]-[0047] explicitly disclose creating an optimized instruction flow stream which *omits the instructions* corresponding to the *overridden*  $C=A+B$  operation (i.e., branching process for selectively performing a specialized path) as opposed to removing the  $C=A+B$  instruction at the decoding stage.

Argument 6 (Brief, page 22): Linden does not teach dynamically detecting a command for which a variable can be limited to a predetermined constant value.

Response to Argument 6: As discussed above, paragraphs [0037]-[0038] and [0041] of Linden explicitly discloses the **dynamic** decoding (i.e., **analysis**) stage going through block of source instructions and *analyzes* the operation codes such as the addition (i.e., **command**) of two numbers (i.e., **parameters**). Specifically in paragraph [0038], Linden explicitly discloses translating the assignment (i.e., command) of Register 3 (i.e., one of the parameters) to the outcome of "Register 1 XOR Register 1" which is always **fixed in a specific state** because clearly it does not matter what the value of variable Register 1 is, the result of XOR-ing a variable and itself is always the **constant 0** (i.e., Boolean false). In other words, the Register 3 is the result (i.e., variable) of the XOR command. But since the parameters of the command XOR are the Register 1 and itself, the result (i.e., variable Register 3) can be limited to the Boolean value False (i.e., constant value Zero), that is to say, the decoding stage simply assigns Register 3 to the constant value Zero instead of repeatedly evaluating the command Register 1 XOR Register 1 (which will always result in the constant value Zero).

Argument 7 (Brief, page 24): Shaylor's copying the code of the target method into the calling method is not the same as or suggestive of detecting a command for which a method call destination can be identified and for which the processing speed can be increased.

Response to Argument 7: First it is respectfully submitted that Shaylor's *target method* should be understood as the method being called (i.e., method call destination) from the calling method. Furthermore, in order to inline the target method into the calling method (i.e., copy the code of the target method into the code of the calling method), it is inherent that the actual calling command/instruction (inside the calling method) be detected so as to determine the exact position for inlining (i.e., inserting the code of the target method). Moreover, in order to determine whether to inline the target code, it is inherent that the target method (i.e., method call destination) be identified as a relatively small method, since the smaller the target method, the more advantage inlining gives because it takes far more time (i.e., slower execution speed) to enter and exit the target method than the actual execution of the target method's content (see at least Shaylor col.2:65-col.3:5).

Argument 8 (Brief, page 25): "inlining cause code size to increase, which effectively teaches away from the claimed steps for increasing processing speed for a command".

Response to Argument 8: As addressed above in Response to Argument 7, the smaller the target method (i.e., method call destination), the more advantage inlining gives because it takes far more time (i.e., slower execution speed) to enter and exit the target method than the actual execution of the target method's content (see at least Shaylor col.2:65-col.3:5). Thus, contrary to Appellants' argument, the inlining of smaller

Art Unit: 2192

target methods increases the execution speed, which is necessary in Shaylor's code optimization.

**(11) Related Proceeding(s) Appendix**

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.



Art Unit: 2192

For the above reasons, it is believed that the rejections should be sustained.

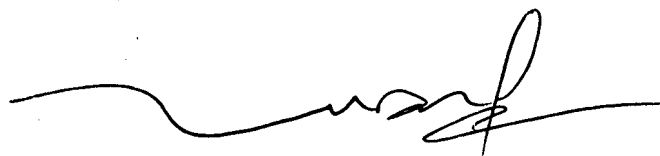
Respectfully submitted,

Chrystine Pham



Conferees:

Tuan Dam, SPE 2192



TUAN DAM  
SUPERVISORY PATENT EXAMINER

Wei Zhen, SPE 2191



WEI ZHEN  
SUPERVISORY PATENT EXAMINER